TD2 Asm Cortex-M3

Pascal Acco pour l'équipe des branquignolles

Résumé

Le but de ce TD est de vous exercer à l'utilisation de sauts conditionnels pour réaliser des structures algorithmiques en peu avancées. Vous serrez aussi amenés à utiliser un pointeur pour parcourir un tableau et effectuer des opérations binaires.

I. Petites questions rapides

A. Arithmétique signée Zorro

Soit les deux programmes suivants

```
zorro
        LDR     R0,   =0xFFFFFFFF
        ADDS    R0,   R0,   #1
        ADC     R1,   R0
```

```
zorro
        LDR     R0,   =0x7FFFFFFF
        ADDS    R0,   R0,   #1
```

Indiquez ce que vaut le registre R_0 et les fanions $ZCNV$ à la fin de chacun des deux programmes. Donnez l'interprétation signée et non–signée des opérandes et des résultats. Comment faire une addition non-signée sur 64 bits ?

B. Joly Jumper

Dans les programmes suivants indiquez le nombre de passages dans la boucle et le point de sortie de chaque boucle.

```
        LDR   R0, = 0XFFFFFFFE
joly
        ADDS R0,   R0,   #1
        BEQ   jumper
        BMI   petittonnerre
        B     joly
```

```
        LDR   R0,   =0x0FFFFFFE
joly
        ADDS        R0, #1
        BCS         jumper
        CMP         R0,   #0x10000000
        BGT         bucephale
        B           joly
```

II. Problème : Calcul de hachage

On se propose de calculer le *hash*[1] d'un mot de passe avec l'algorithme ELFhash souvent utiisé dans les systèmes Linux[2]

Il est courant de vérifier l'authenticité d'un mot de passe en comparant le hachage du texte saisi avec le hachage du mot de passe calculé et stocké lors de son initialisation. On évite ainsi de stocker le texte clair dans le programme sensé contrôler un accès.

On peut aussi vérifier l'authenticité d'un long flot de données (plusieurs Mo) en comparant le hash du texte authentique (de courte longueur 32 à 256 bits) avec celui du texte reçu. Dans ce cas, le calcul du hash peut être long : on fait alors appel à des routines optimisées.

1. On nomme fonction de hachage une fonction particulière qui, à partir d'une donnée fournie en entrée, calcule une empreinte servant à identifier rapidement, bien qu'incomplètement, la donnée initiale. Les fonctions de hachage sont utilisées en informatique et en cryptographie.
2. Nous choisissons le GNU bien que l'Afghan soit réputé pour produire du très bon *hash*.

Question 1 : Compiler un algorithme.

Saurez-vous faire mieux en assembleur qu'un compilateur dans le cas de l'algorithme de hachage suivant :

```
unsigned int ELFHash(char* str, unsigned int len)
{
    unsigned int hash = 0;
    unsigned int x    = 0;
    unsigned int i    = 0;

    for(i = 0; i < len; str++, i++)
    {
            hash = (hash << 4) + (*str);

            if((x = hash & 0xF0000000L) != 0)
            {
                    hash ^= (x >> 24);
            }

            hash &= ~x;
    }
    return hash;
}
```

Traduisez cet algorithme en assembleur en indiquant l'association entre registres et variables du programme. Tentez ensuite d'en optimiser le temps d'exécution.

Question 2 : Passage de paramètres.

Écrivez un module assembleur contenant la procédure `HELFHash` traduite. Cette procédure pourra être appelée en C avec le même prototype.

Pour cela il suffit d'utiliser R0 et R1 comme paramètre d'entrée de la fonction et de renvoyer le résultat dans R0. L'appel de la fonction assembleur ne peut altérer que les registre R0 à R3 et PC bien sûr.

Question 3 : Faire un appel.

Écrire un module appelant la fonction HELHash (écrite dans un autre module) calculant le hash de "Salut les geeks".